SciSpot: Scientific Computing On Transient Cloud Servers

Paper # 105

ABSTRACT

Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers can be used to lower the costs of running applications on the cloud. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient servers, and present SciSpot, a software framework that enables low-cost scientific computing on the cloud. SciSpot deploys applications on Google Cloud preemptible virtual machines, and introduces the first empirical and analytical model for their preemptions.

SciSpot's design is guided by our observation that many scientific computing applications (such as simulations) are deployed as "bag" of jobs, which represent multiple instantiations of the same computation with different physical and computational parameters. For a bag of jobs, SciSpot finds the optimal transient server on-the-fly, by taking into account the price, performance, and preemption rates of different servers. SciSpot reduces costs by 5× compared to conventional cloud deployments, and reduces makespans by up to 10× compared to conventional high performance computing clusters.

1 INTRODUCTION

Scientific computing applications are a crucial component in the advancement of science and engineering, and play an important role in the analysis and processing of data, and understanding and modeling natural processes. These applications are usually parallelized and require a large amount of computing resources (1000s of CPU cores), and are deployed on large, dedicated high performance computing (HPC) infrastructure such as super computers.

Increasingly, cloud computing platforms have begun to supplement and complement conventional HPC infrastructure to meet the large computing and storage requirements of scientific computing applications [58]. Applications can benefit from the multiple benefits of public clouds: on-demand resource allocation, convenient pay-as-you-go pricing models, ease of provisioning and deployment, and near-instantaneous elastic scaling.

However, large deployment costs remains a concern for scientific computing. Cloud deployment costs can be reduced through the use of *transient* computing resources (i.e., VMs) that can be unilaterally revoked and preempted by the cloud provider. Due to their volatile nature, transient VMs such as Amazon EC2 Spot instances [12], Google Preemptible VMs [7], and Azure Batch VMs [3], are offered at steeply discounted rates ranging from 50 to 90%. Although using transient resources can drastically reduce computing costs, their preemptible nature results in frequent job failures, and thus reduces their viability and usability.

Deploying scientific applications on cloud platforms presents multiple challenges due to the *fundamental* differences with conventional HPC clusters. HPC resource management frameworks such as Slurm [9] and Torque [11] typically assume static, homogeneous clusters, and are not cognizant of transient availability and the plethora of choices in cloud VM sizes and pricing. Crucially, optimizing for *cost* in addition to performance, is an important objective in cloud deployments.

In this paper, we present SciSpot, a framework that optimizes the deployment of scientific computing applications on transient cloud servers. SciSpot introduces and incorporates policies for addressing the heterogeneity and preemptibility of transient cloud VMs, and is the *first* framework for transient computing that is not limited to Amazon EC2 spot instances, whose distinctive characteristics are inapplicable to other transient cloud VMs.

SciSpot can run scientific applications on transient Google Preemptible VMs, which raises a number of hitherto unexplored challenges. First, the preemption characteristics of these VMs, such as their time to preemption, is unspecified by the cloud provider, requiring extensive empirical study. Second, Google Preemptible VMs have a *maximum* lifetime of 24 hours, which results in preemptions being constrained within a 24 hour window. Understanding and modeling these preemptions requires new fundamental techniques, since existing reliability models (such as exponential and Weibull probability distributions for modeling lifespans) cannot meet the hard temporal constraints.

To address these challenges and to develop cost and runtime minimizing resource allocation policies, we conduct large-scale experiments with over a thousand preemption events of Google Preemptible VMs, and develop a new reliability model for constrained preemption. This model introduces a new failure probability distribution, that allows us to mitigate preemptions, and predict expected running times and costs of different scientific computing applications.

, ,
.

While several approaches for mitigating preemptions of transient cloud VMs have been proposed, they all rely on preemption information provided by Amazon EC2 spot prices. Transiency-specific frameworks that use techniques such as VM migration [64], checkpointing [54, 62], diversification [63], *all* use price-signals to model the availability and preemption rates of spot instances. However, these pricing-based models are not generalizable to other transient VMs having a flat price (such as Google's or Azure's offerings), or even across time, as Amazon has recently changed their pricing models [13] and the applicability of price-based approaches remains uncertain [14]. Unlike prior work on transient computing, SciSpot's empirical preemption model allows effective use of a wider range of transient cloud resources (such as Google Preemptible VMs).

SciSpot abstracts typical scientific computing workloads and workflows into a new unit of execution, which we call as a "bag of jobs". These bags of jobs, ubiquitous in scientific computing, represent multiple instantiations of the same application launched with possibly different physical and computational parameters. Collectively, a bag of jobs can be used to "sweep" or search across a multi-dimensional parameter space to isolate the set of desired parameters associated with the scientific computation model. Bags of jobs also help in the use of machine learning (ML) to enhance scientific computational methods [18, 22, 30, 52, 53, 60, 72], when a collection of jobs with different parameters are launched to train and test ML models.

Treating bag of jobs as a fundamental unit of computation enables SciSpot to select the "best" server configuration for a given application that minimizes the costs, running time, and preemption likelihood. SciSpot can find the optimal allocation on-the-fly by exploring different servers for initial jobs and running the remainder of the jobs on the optimal server configuration. For a bag of jobs, it is not necessary, or sufficient, to execute an individual job in timely manner—instead, we can selectively restart failed jobs in order to complete the necessary, desired subset of jobs in a bag. In contrast, prior work on transiency-mitigation has mostly focused on fault-tolerance and resource management for a *single* job or application [54, 62, 63, 67].

Our empirical preemption model, cost-optimizing server-selection policies, and the bag of jobs abstraction are implemented as part of the SciSpot framework, and we make the following contributions:

(1) We develop a new analytical model based on a large-scale, first-of-its-kind empirical study of lifetimes of Google preemptible VMs for understanding and characterizing their preemption dynamics. Our model captures the key effects resulting from the 24 hour lifetime constraint associated with these VMs, and enables accurate prediction of expected running times and costs of different scientific computing applications.
(2) In order to select the optimal VM for an application, from the plethora of choices offered by cloud providers, we develop a transient VM selection policy that minimizes the cost of running applications. Our search based policy selects a transient VM based on it's cost, performance, and preemption rate.
(3) We implement all our policies as part of a new framework, SciSpot, and evaluate the cost and performance of different representative scientific computing applications on the Google Cloud Platform. Compared to conventional cloud deployments, SciSpot can reduce costs of running bags of jobs by more than 5×, and when compared to dedicated HPC clusters, it can reduce the total turnaround time by up to an order of magnitude.

2 BACKGROUND

We now give an overview of transient cloud computing, and motivate the need for the bag of jobs abstraction in scientific computing workflows.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an "on-demand" basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variation. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [25, 71].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers with *transient* availability, which can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [2], Google Preemptible VMs [7], and Azure batch VMs [3], have become popular in recent years due to their discounted prices, which can be 7-10× lower than conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as Packet [8] and Alibaba [1] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [61, 64]. Preemptions are akin to fail-stop failures, and result in loss of the application's memory and disk state, leading to downtimes for interactive applications such as web services,

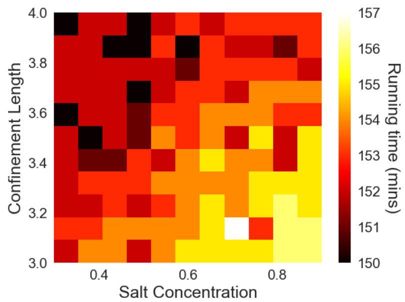

Figure 1: Running times of the Nanoconfinement application with different sets of parameters (particle size and length) show little variation.

and poor throughput for long-running batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [54, 62, 67] and resource management techniques [63] to ameliorate the effects of preemptions. The effect of preemptions is dependent on a combination of application resource and fault model, and mitigating preemptions for different applications remains an active research area [45].

Past work on transient computing has almost exclusively focused on the Amazon EC2 spot market, which has a radically different preemption model and dynamics compared to other transient cloud servers such as those offered by Google Cloud and Azure. In contrast, SciSpot can run applications on Google Preemptible VMs that have fixed price and *unknown* preemption rates. Thus due to the lack of any historical preemption information and the fundamental differences in the preemption characteristics, techniques such as historical price-based bidding strategies [82], price-based checkpointing [36, 54, 67], and diversification [14, 63], are unfortunately largely inapplicable. To address these shortcomings, we develop an empirical model of preemptions in Section 3. This model informs our policies for minimizing cost and running time of scientific computing workloads that are launched as bags of jobs, as described next.

2.2 Bag of Jobs in Scientific Computing

The typical workflow associated with most scientific computing applications, often involves evaluating a computational model across a wide range of physical and computational parameters. For instance, constructing and calibrating a molecular dynamics application (such as Nanoconfinement [46]), usually involves running a simulation with different physical parameters such as characteristic sizes and interaction potentials, as well as computational parameters such as simulation timesteps. Each of these parameters can take a wide range of values, resulting in a large number of combinations which must be evaluated by invoking the application multiple times (also known as a parameter sweep). Since each computational job explores a single combination of parameters, this results in executing a "bag of jobs", with each job in the bag running the same application, but with possibly different parameters. Bags of jobs are analogous to, but distinct from the bag of *tasks* design that is for embarrassingly parallel applications [24]. In contrast, the bags of jobs abstraction is independent of the application design.

The bag of jobs execution model is pervasive in scientific computing and applicable in many contexts. In addition to exploratory parameter sweeps, bags of jobs also result from running the application a large number of times to account for the model or computational stochasticity, and can be used to obtain tighter confidence intervals. Increasingly, bags of jobs also arise in the emerging research that combines statistical machine learning (ML) techniques and scientific simulations [18, 22, 60]. For instance, large bags of jobs are run to provide the necessary training and testing data for learning statistical models (such as neural networks) that are then used to improve the efficacy of the simulations [46].

The bag of jobs execution model has multiple characteristics, that give rise to unique challenges and opportunities when deploying them on transient cloud servers. Since bags of jobs require a large amount of computing resources, deploying them on the cloud can result in high overall costs, thus requiring policies for minimizing the cost and overall running time. The similarity in execution characteristics of jobs (such as their running times and parallel speedup) allows for bag-wide optimizations (Figure 1). And last, treating entire bags of jobs as an execution unit, instead of individual jobs, can allow us to use partial redundancy between jobs and reduce the fault-tolerance overhead to mitigate transient server preemptions.

3 MODELING PREEMPTION DYNAMICS

To measure and improve the performance of applications running on transient cloud servers, it is critical to understand the nature and dynamics of their preemptions. In this section, we present empirical analysis of preemptions of Google Preemptible VMs, and develop new probabilistic models of their preemption rates.

3.1 Need For Empirical Preemption Models

Launched in 2009, Amazon's EC2 spot instances are the first example of transient cloud servers. The preemptions of EC2 spot instances are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server is preempted [19].

Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" (MTTF) of the spot instances. Publicly available[1] historical spot prices have been used to characterize and model spot instance preemptions [64, 82]. For example, past work has analyzed spot prices and shown that the MTTFs of spot instances of different hardware configurations and geographical zones range from a few hours to a few days [59, 61, 75].

However, pricing based models and the policies based on them are at risk when there are changes in the cloud provider's pricing mechanisms and policies. For instance, Amazon has recently changed the preemption characteristics of spot instances, and servers are now preempted even if the spot price is below the maximum price [13, 14]. Thus, spot prices are no longer a completely reliable indicator of preemptions, and preemptions can no longer be inferred from looking at prices alone. Therefore, new techniques are required to model preemption dynamics that can supplement the earlier price-based approaches, and we develop these techniques next.

3.2 Empirical Study Of Preemptions In Google Cloud

The preemptions of transient servers need not be related to their price. For example, Google's Preemptible VMs and Azure Batch VMs have a *fixed* price relative to their non-preemptible counterparts. In such cases, price-based models are inadequate, and other approaches to understand preemptions are required.

This task is further complicated by the fact that these cloud operators (Google and Microsoft) do not currently provide any information about preemption characteristics. Thus, relatively little is known about the preemptions (and hence the performance) of these transient VMs. To understand preemption dynamics of transient servers, we conduct a large-scale empirical measurement study. We launched more than 1,500 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (i.e., their useful lifetime).[2]

A sample of over 100 such preemption events are shown in Figure 2, which shows cumulative distribution function (CDF) of the VM lifetimes of the `n1-highcpu-16` VM in the `us-east1-b` zone. Note that the cloud operator (Google) caps the *maximum* lifetime of the VM to 24 hours, and all the VMs are preempted before that limit.

Observation 1: *The lifetimes of VMs are not uniformly distributed, but have three distinct phases.*

In the first (initial) phase, characterized by VM lifetime $t \in [0, 3]$ hours, we observe that many VMs are quickly preempted after they are launched, and thus have a steep rate of failure (derivative of the CDF) initially. In the second phase,

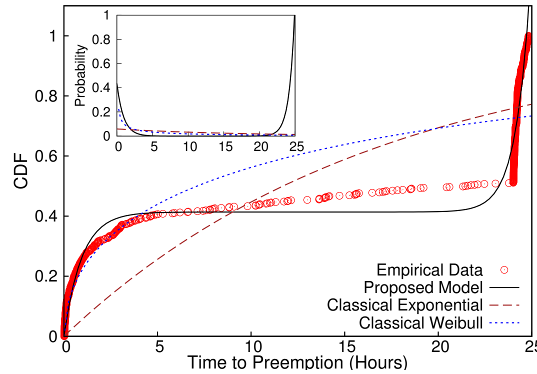

Figure 2: CDF of lifetimes of Google Preemptible VMs. Our proposed distribution for modeling the constrained preemption dynamics provides better fits to the empirical data compared to the conventional exponential and the Weibull distributions. Inset shows the probability of failure as a function of time for the three distributions.

VMs that survive past 3 hours enjoy a relatively low preemption rate over a relatively broad range of lifetime (characterized by the slowly rising CDF in Figure 2). The third and final phase exhibits a steep increase in the number of preemptions as the preemption deadline of 24 hours approaches. The overall rate of preemptions is "bathtub" shaped as shown in the inset of Figure 2.

Observation 2: *The preemption behavior, imposed by the constraint of the small 24 hour lifetime, is substantially different from conventional failure characteristics of hardware components and even EC2 spot instances.*

In "classical" reliability analysis, the rate of failure usually follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. Figure 2 shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. The classic exponential distribution is unable to model the observed preemption behavior because it assumes that the rate of preemptions is independent of the lifetime of the VMs, i.e., the preemptions are *memoryless*. This assumption breaks down when there is a fixed upper bound on the lifetime, as is the case for Google Preemptible VMs, and the conventional approach becomes insufficient to model this constrained preemption dynamics.

3.3 Modeling Constrained Preemption Dynamics

We now develop an analytical model for preemption dynamics that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations. To describe failures (preemptions) that are not memoryless (i.e., increasing or decreasing failure rate

[1] Amazon posts Spot prices of 3 months, and researchers have been collecting these prices since 2010 [43].

[2] We will release the complete preemption dataset for further analysis.

over time), the classic Weibull distribution with CDF $F(t) = 1 - e^{-(\lambda t)^k}$ is often employed. However, we find that the Weibull distribution is also unable to fit the empirical data (Figure 2).

The non-trivial bathtub-shaped failure rate of Google preemptible VMs (Figure 2) requires models that capture the sudden onset of the rise in preemptions near the deadline. Our new model, informed by the cumulative distribution of lifetimes that has multiple distinct temporal phases, addresses this need. The key assumption underlying our model is the presence of two distinct failure processes. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the high rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed to be characterized by an exponential term that captures the sharp rise in preemptions that results from this constrained lifetime.

Based on these observations, we propose the following general form for the CDF:

$$\boxed{\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right)} \tag{1}$$

where t is the time to preemption, $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase, b denotes the time that characterizes "activation" of the final phase where preemptions occur at a very high rate, and A is a scaling constant. In practice, typical fit values yield $b \approx 24$ hours, and $\tau_2 \approx 1$ hour, which ensures that our proposed distribution meets the initial condition $\mathscr{F}(0) \approx 0$.

For most of its life, a VM sees failures according to the classic exponential distribution with a rate of failure equal to $1/\tau_1$ – this behavior is captured by the $1 - e^{-t/\tau_1}$ term in Equation 1. As VMs get closer to their maximum lifetime imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high rate $1/\tau_2$, which is captured by the second exponential term, $e^{(t-b)/\tau_2}$ of Equation 1. Shifting the argument (t) of this term by b ensures that the exponential reclamation is only applicable near the end of the VM's maximum lifetime and does not dominate over the entire temporal range.

The analytical model and the associated distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data (Figure 2) and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters can be obtained for a given empirical CDF using least squares function fitting methods.

Further, the failure or preemption rate can be derived from this CDF as:

$$p(t) = \frac{d\mathscr{F}(t)}{dt} = A\left(\frac{1}{\tau_1}e^{-t/\tau_1} + \frac{1}{\tau_2}e^{\frac{t-b}{\tau_2}}\right). \tag{2}$$

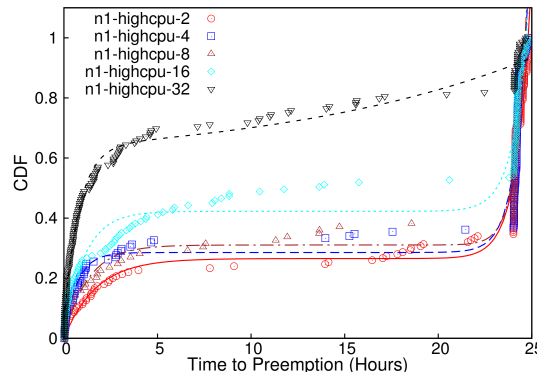

Figure 3: Preemption characteristics of different VM types. Larger VMs are more likely to be preempted.

$p(t)$ vs. t yields a bathtub type failure rate function for the associated fit parameters (inset of Figure 2). In the next section, we use this analytical model for optimizing cloud resource selection to run scientific computing applications at low cost and shorter running (turnaround) times.

In the absence of any prior work on constrained preemption dynamics, our aim is to provide an interpretable model with a minimal number of parameters, that provides a sufficiently accurate characterization of observed preemptions data. Further generalization of this model to include more failure processes would introduce more parameters and reduce the generalization power. Exploring other approaches of modeling bathtub-type failure rates (e.g., exponential Weibull distributions) [26, 57] is part of our future work.

3.4 Analysis of Preemption Dynamics

In general, the preemption dynamics of a VM is determined by the supply and demand of VMs of that *particular* type. Thus, the preemption characteristics of VMs of different sizes, and running in different geographical zones, are different. Figure 3 shows the preemption data from five different types of VMs in the Google Cloud `n1-highcpu-{2,4,8,16,32}`, where the number indicates the number of CPUs. All VMs are running in the `us-central1-c` zone. We also show the associated CDFs ($\mathscr{F}$) computed using the proposed model.

Observation 3: *CDFs obtained using our model can capture the preemption dynamics of different VM types.*

For the smallest four VM sizes (2, 4, 8, 16), we find that the initial rate of preemptions ($1/\tau_1 \in [0.5, 1.5]$ hr^{-1}) is typically smaller than the final rate of preemptions ($1/\tau_2 \in [1.28, 1.72]$ hr^{-1}), and the activation time for the final phase $b \in [24, 24.5]$ hours. We note the distinct behavior of the analytical CDF for VMs of size 32, where the fit does not reproduce the final rise accurately but captures the slightly faster increase in preemptions during the middle phase better. These plots also illustrate a deficiency in our model, whereby the boundary condition of $\mathscr{F} = 1$ for $t = 24$ hours is not strictly imposed.

Observation 4: *Larger VMs have a higher rate of failure.* Larger VMs require more computational resources (such as CPU and memory), and when the supply of resources is low, the cloud operator can reclaim a large amount of resources by preempting larger VMs. This observed behavior aligns with the guidelines for using preemptible VMs that suggests the use of smaller VMs when possible [7].

Our analytical model also helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM of type i, as:

$$E[L_i] = \int_0^{24} tp_i(t)\, dt = -A(t+\tau_1)e^{-t/\tau_1} + A(t-\tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^{24} \quad (3)$$

where $p_i(t)$ is the rate of preemptions of VMs of type i (Equation 2). We use the analytically derived expected lifetimes of VMs of different types in SciSpot when selecting the "best" VM type for a given bag of jobs. This server selection is a key part of SciSpot design, which we describe next.

4 SCISPOT DESIGN

SciSpot is a general-purpose software framework for running scientific computing applications on low-cost transient cloud servers. It incorporates policies and mechanisms for generating, deploying, orchestrating, and monitoring bags of jobs on cloud servers. Specifically, it runs a bag of jobs defined by these parameters:

Bag of job = {$\mathcal{A}$: Application to execute,
n: Number of jobs,
m: Minimum number of jobs to finish,
π: Generator function for job parameters,
$\mathcal{R}$: Computing resources per job}

SciSpot seeks to minimize the cost and running time of bags of jobs of scientific computing applications. SciSpot's cost and time minimizing policies for running bags of jobs are based on empirical and analytical models of the cost and preemption dynamics of transient cloud servers, which we present in the next section.

SciSpot is designed as a framework that increases the usability and viability of transient cloud servers for scientific computing applications, and provides a simple user interface to allow users to deploy their applications with minimum workflow changes. Most scientific computing applications are deployed on HPC clusters that have a batch scheduler such as Slurm [9] or Torque [11], and SciSpot integrates with these schedulers (e.g., Slurm) to provide the same interface to applications. As shown in Figure 4, SciSpot creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the various policies described in the rest of this section.

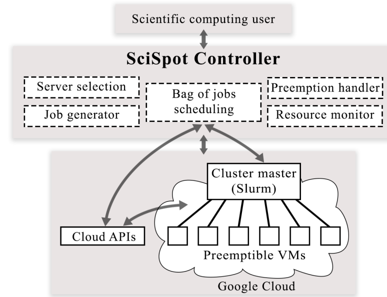

Figure 4: SciSpot architecture and system components.

High-level workflow: When a user wishes to run a bag of jobs, SciSpot handles the provisioning of a cluster of transient cloud servers. In addition, SciSpot deals with the scheduling and monitoring of the bag of jobs, and with VM preemptions. Execution of a bag of jobs proceeds in two phases. In the first phase, SciSpot selects the "right" cluster configuration for a given application through a cost-minimizing exploration-based search policy, described in Section 4.1. In the second phase, SciSpot proceeds to run the remaining jobs in the bag on the optimal cluster configuration.

4.1 Server Selection

4.1.1 Why Server Selection is Necessary. Before deploying any application on the transient cloud servers, the appropriate cloud server for the application must be selected. Cloud platforms offer a large range of servers (VMs) with different resource configurations (such as the number of CPU cores, memory size, I/O bandwidths, etc.). *Importantly, different server configurations have different cost, performance, and preemption characteristics.*

Even if we assume that the total amount of resources to be allocated to a job is fixed, there are multiple *cluster configurations* to satisfy the allocation with the large number of available server types. Server selection is especially important for parallel applications, because although the total amount of resources in each cluster configuration is constant, the resources are distributed differently—i.e., a job can run on either 2 VMs with 32 CPUs each, or a single 64 CPU VM. Since the performance of parallel applications is particularly sensitive to their communication overheads, different cluster configurations may yield different job running times. For instance, a smaller cluster with large VMs will result in lower inter-VM communication, and thus shorter running times.

However, the performance of an application is also affected by the preemptions of transient servers. Since preemptions are essentially fail-stop failures, synchronous parallel applications (such as those using MPI) are forced to abort, and completing the job requires restarting it. Thus, frequent preemptions can increase the overall turnaround time of a job.

4.1.2 Server Selection Policy. Having provided the motivation and tradeoffs in server selection, we now describe the

SciSpot's server selection policy. Given an application and a bag of jobs, SciSpot "explores" and searches for the right server type by minimizing the expected cost of running the job. Since jobs in a bag have similar execution characteristics, optimizing server selection for an individual job also translates to the entire bag.

SciSpot allows the users to specify the total amount of resources required per job, which we denote by $\mathcal{R}$. For example, $\mathcal{R}$ can be the total number of CPU cores. It first determines the search space, which is the space of all cluster configurations (i, n_i) such that $r_i n_i = \mathcal{R}$, where r_i is the resource size of a VM of type i (e.g., number of CPUs), and n_i is the number of VMs of that type. Based on the constraint, the number of servers of type i required is $n_i = \mathcal{R}/r_i$.

Each cluster configuration yields different application performance, preemption overhead, and cost. The aim is to find the lowest-cost configuration (i, n_i) for a given application. The server selection policy runs the application on different cluster configurations to determine the base running time (in the absence of preemptions), which is denoted by $T_{(i,n_i)}$. It then combines the empirical running time with a cost model, to estimate the expected cost of running the application.

4.1.3 *Server Cost Model.*

Since server selection involves a tradeoff between cost, performance, and preemptions, we develop a model that allows us to optimize the resource allocation and pick the best VM type that minimizes the expected cost of running an application on SciSpot.

Let us assume that the cloud provider offers N server types, with the price (per unit time) of a server type equal to c_i. The overall expected cost of running a job can then be expressed as follows:

$$E[C_{(i,n_i)}] = n_i \times c_i \times E[\mathcal{T}_{(i,n_i)}]. \tag{4}$$

Here, $E[\mathcal{T}_{(i,n_i)}]$ denotes the expected turnaround time of the job (accounting for preemptions) on n_i servers of type i. This turnaround time depends on whether the job needs to be recomputed because of preemptions, and is expressed as:

$$E[\mathcal{T}_{(i,n_i)}] = T_{(i,n_i)} + E[\text{Recomputation Time}]. \tag{5}$$

Here, $T_{(i,n_i)}$ is the base running time of a job without preemptions, which we obtain empirically as explained in the previous subsection. Since jobs have to be rerun when they fail due to preemptions, the recomputation time is:

$$E[\text{Recomputation Time}] = \frac{T_{(i,n_i)}}{2} \times P(\text{at least one preemption}) \tag{6}$$

Our expression of the recomputation time is based on the common assumption that jobs will fail at the half-way mark on average [21, 27]. The probability that at least one VM out of n_i will be preempted during the job execution can be expressed as:

$$P(\text{at least one preemption}) = 1 - P(\text{no preemptions}) \tag{7}$$

$$= 1 - \left(1 - P\left(i, T_{(i,n_i)}\right)\right)^{n_i}. \tag{8}$$

Here, $P(i, T_{(i,n_i)})$ denotes the probability of a preemption of a VM of type i when a job of duration $T_{(i,n_i)}$ runs on it. It depends on the type of server, and its associated expected lifetime, and is defined as:

$$P\left(i, T_{(i,n_i)}\right) = \min\left(\frac{T_{(i,n_i)}}{E[L_i]}, 1\right), \tag{9}$$

where $E[L_i]$ is the expected lifetime of the VM of type i extracted using the preemption model (Equation 3). We also assume that the running time of *individual* jobs in a bag (T), will be smaller than the expected lifetime of the VMs, otherwise we will see no forward progress since the jobs will always be preempted before completion. This is a safe assumption, since more than 90% HPC jobs are less than 2 hours long (Figure 7 inset), and the expected lifetime of transient VMs is more than 10 hours. This restriction only applies to individual jobs—SciSpot can run large bags of jobs even if their total running time exceeds the VM lifetime by replenishing preempted VMs.

Combining all the equations, we see that the expected cost $E[C_{(i,n_i)}]$ is higher for larger number of servers (high n_i), while it is reduced if the expected lifetime of the VM is larger (high $E[L_i]$). Thus, if we select VMs of smaller size, we will require more of them (higher n_i), and this cluster configuration will have a larger probability of failure and thus higher running times and costs. However, there is a tradeoff: selecting larger VMs results in smaller n_i, but larger VMs have higher preemption probability (Section 3.4).

To limit the search space, we observe that since most scientific computing applications are CPU bound, we only need to consider VMs meant for CPU-bound workloads, such as `highcpu` VMs in Google Cloud and the `cc` family in Amazon EC2. For example, the Google cloud offers a total of 7 `highcpu` server types with 1, 2, 4, 8, 16, 32, and 64 CPU's—yielding a small upper bound on the number of configurations to search. Furthermore, a large cluster of small servers is suboptimal for most applications (except those that are completely embarrassingly parallel and have no communication). SciSpot thus explores VMs in descending order of their size and ignores exploring the small VMs (with 1 and 2 CPUs)—reducing the search space even further.

4.2 Scheduling a Bag of Jobs

Once the right cluster configuration for a job has been determined, SciSpot then proceeds to run the remaining jobs in the bag on the cluster of VMs found through the exploration.

Upon job completion, we run the next job in the bag on the existing cluster of preemptible VMs, with job parameters obtained using the generator function $\pi.next()$. This policy is based on our preemption dynamics model which shows that preemption rates have a bathtub shape. Thus, jobs launched on "stable" VMs that have been running for a few hours, face low likelihood of failures—thereby reducing the number of job failures for the entire bag.

SciSpot also allows users to specify a deadline for bag completion, which we use to compute the number of *jobs* to execute in parallel. If the deadline specified is D, then the number of parallel jobs is $k = D/E[\mathcal{T}]$. Thus if the exploration phase recommends n_i VMs, then we launch a cluster of $k \times n_i$ VMs, with each job executing on n_i VMs. For this calculation, we assume that the running time of different jobs in a bag will largely be similar (as illustrated in Figure 1), but this is not a correctness requirement. Due to the stochasticity in job running times and VM lifetimes, SciSpot only meets the deadline in a "best effort" manner, and does not guarantee makespan constraints.

When a job fails due to VM preemption, SciSpot replenishes the cluster by launching replacement VMs and resubmits the job. Jobs are restarted from a checkpoint if available. We do not restart failed jobs if we complete the minimum number of jobs in the bag. Due to high demand, preemptible VMs of the chosen type may not be available. In such cases, SciSpot runs in a "degraded" mode—jobs are either run on a smaller number of VMs, or are run on VMs of a different size that are available but may have suboptimal cost.

Checkpointing Policy. When applicable, SciSpot resumes jobs from the latest checkpoint performed using tools such as DMTCP [17]. Since checkpointing also increases the running time of the job, the checkpoint interval must be carefully computed. The classic Young-Daly periodic checkpointing interval [27] is only applicable when failures follow an exponential distribution, which we have shown is not true in the case of Google Preemptible VMs (Figure 2). Our analytical model for preemptions permits advanced, non-periodic checkpointing intervals that can be computed using a dynamic programming approach similar to [21].

5 SCISPOT IMPLEMENTATION

SciSpot is implemented as a light-weight, extensible framework that makes it convenient and cheap to run scientific computing applications in the cloud. We have implemented the SciSpot prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [6]. SciSpot is implemented as a centralized controller, which implements the VM selection and job scheduling policies described in Section 4. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit bags of jobs to the controller via the HTTP API, which then launches and maintains a cluster of cloud VMs, and maintains the job queue and metadata in a local database. For improving usability, SciSpot can automatically generate parameter combinations for a given bag size—based on a user-provided json file with ranges and constraints for each parameter.

SciSpot integrates, and interfaces with two primary services. First, it uses the Google cloud API [5] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm [9] or Torque [11], to which it submits jobs. SciSpot uses the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. The Slurm master node runs on a small, 2 CPU non-preemptible VM, which is shared by all applications and users. SciSpot monitors job completions and failures (due to VM preemptions) through the use of Slurm call-backs, which issue HTTP requests back to the SciSpot controller.

6 EXPERIMENTAL EVALUATION

In this section, we present empirical and analytical evaluation of the performance and cost of SciSpot with different scientific computing workloads and scales. Our evaluation consists of empirical analysis, as well as model-driven simulations for analyzing and comparing SciSpot behavior under different preemption and application dynamics.

Environment and Workloads: All our empirical evaluation is conducted on the Google Public Cloud, and with these representative scientific computing applications:

Nanoconfinement. The nanoconfinement application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [41, 47].

Shapes. The Shapes application runs an MD-based optimization dynamics to predict the optimal shape of deformable, charged nanoparticles [40, 44].

LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is a popular benchmark for hydrodynamics simulations of continuum material models [48, 49].

These examples are representative of typical scientific computing applications in the broad domain of physics, materials science, and chemical engineering. These three examples are implemented as parallel programs that use OpenMP and MPI parallel computing techniques. The first two are used in nanoscale materials research [38–41, 44, 65] and LULESH is a widely used benchmark [48, 49]. All applications are run with default parameters unless otherwise stated.

All applications use OpenMPI v2.1.1, are deployed on Slurm v0.4.3 and 64-bit Ubuntu 18.04, and run on Google Cloud VMs with x86-64 Intel Haswell CPUs. For completeness and to guard against concerns about poor cloud performance relative to HPC clusters [20, 32, 37, 55, 80], we benchmarked the Nanoconfinement application on the Big Red II cluster [4]. When run on 4 nodes with 16 CPUs each,

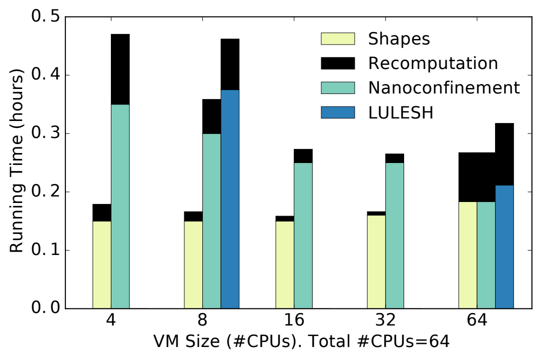

Figure 5: Running times of applications on different VMs. The total number of CPUs is 64, yielding different number of VMs in each case. We see different tradeoffs in the base running times and recomputation times.

the application takes 1140 seconds on Big Red II vs 850 seconds on SciSpot. We attribute the 20% improvement with SciSpot to the newer CPUs on Google Cloud (Intel Haswell vs. older 2012-era AMD Opterons in Big Red II).

6.1 SciSpot Performance and Cost

6.1.1 Impact of server exploration. As described in Section 4, applications can be deployed on multiple types of VMs in the cloud, with each VM type having a different "size". In our evaluation of parallel scientific computing applications that are CPU intensive, we are primarily interested in the number of CPUs in a VM.

When an application (i.e., bag of jobs) requests a total number of CPUs to run each of its jobs, SciSpot first runs its exploration phase to find the "right" VM for the application. SciSpot searches for the VM that minimizes the total expected cost $E[C_{(i,n_i)}]$ of running the application, and this depends on several factors such as the parallel structure of the application, the preemption probability and the associated job recomputation time, and the price of the VM.

Thus, even if the *total* amount of resources (i.e., number of CPUs) per job is held constant, the total running time (i.e., turnaround time) of an application depends on the choice of the VM type (i), and the associated number of VMs (n_i) required to meet the allocation constraint (Section 4.1.3). With preemptible instances, the total running time of a job is composed of two factors: the "base" running time of the job without any preemptions ($T_{(i,n_i)}$), and the expected recomputation time which depends on the probability of job failure (Equation 6).

Figure 5 shows the running times of the Nanoconfinement, Shapes, and LULESH applications, when they are deployed on different VM sizes. In all cases, the total number of CPUs per job is set to 64, and thus the different VM sizes yield different cluster sizes (e.g., 16 VMs with 4 CPUs or 32 VMs with 2 CPUs).

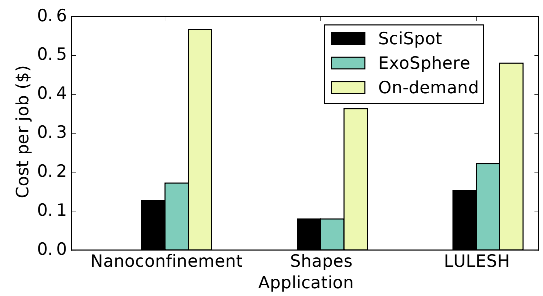

Figure 6: SciSpot's use of preemptible VMs can reduce costs by up to 5× compared to conventional cloud deployments, and 20% compared to the state of the art EC2 spot instance selection (ExoSphere [63]).

For the Nanoconfinement and LULESH applications, we observe that the base running times (without preemptions) reduce when moving to larger VMs, because this entails lower communication costs. For Nanoconfinement, the running time on the "best" VM (i.e., with 32 CPUs) is nearly 40% lower as compared to the worst case. On the other hand, the Shapes application can scale to a larger number of VMs without any significant communication overheads, and does not see any significant change in its running time.

Figure 5 also shows the expected turnaround time $E[\mathcal{T}_{(i,n_i)}]$, that is obtained by adding the the expected recomputation time, which depends on the expected lifetimes of the VM and the number of VMs, and is computed using the cost model introduced in Section 4.1.3. While selecting larger VMs may reduce communication overheads and thus improve performance, it is not an adequate policy in the case of preemptible VMs, since the preemptions can significantly increase the turnaround time. We can observe this in the case of Nanoconfinement application when deployed on a 64 CPU VM—even though the base running time is lower compared to deploying the application on 2x32-CPU VMs, the recomputation time on the 64 CPU VM is almost 4× higher due to the much lower expected lifetime of the larger VMs. Thus, on preemptible servers, there is a tradeoff between the base running time which only considers parallelization overheads, and the recomputation time. By considering *both* these factors, SciSpot's server selection policy can select the best VM for an application.

Result: *SciSpot's server selection, by considering both the base running time and recomputation time, can improve performance by up to 40% , and can keep the increase in running time due to recomputation to less than 5%.*

6.1.2 Cost. The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. Figure 6 compares the cost of running different applications with different cloud VM deployments. SciSpot, which uses both cost-minimizing server selection, and preemptible VMs, results in significantly lower costs across the board, even when accounting for preemptions and recomputations. We also compare against ExoSphere [63], a state of the art system for transient server selection. Unlike SciSpot, ExoSphere only considers server price and preemption rates, and does *not* consider application performance when selecting servers, and thus is unable to select the best server. Since the Google `highcpu` VMs have the same price per CPU, ExoSphere picks an arbitrary "median" VM to break ties, which may not necessarily yield the lowest running times. This results in 20% cost increase over SciSpot.

Result: *SciSpot reduces computing costs by up to 5× compared to conventional on-demand cloud deployments.*

6.1.3 Comparison with HPC Overhead. Scientific computing applications are typically run on large-scale HPC clusters, where different performance and cost dynamics apply. While there are hardware differences between cloud VMs and HPC clusters that can contribute to performance differences, we are interested in the performance "overheads". In the case of SciSpot, the job failures and recomputations increase the job turnaround time, and are thus the main source of overhead.

On HPC clusters, jobs enjoy significantly lower recomputation probability, since the hardware on these clusters has MTTFs in the range of years to centuries [28]. However, we emphasize that there exist *other* sources of performance overheads in HPC clusters. In particular, since HPC clusters have high resource utilization, they also have significant *waiting* times. On the other hand, cloud resource utilization is low [71] and there is usually no need to wait for resources, which is why transient servers exist in the first place.

Thus, we compare the performance overhead due to preemptions for SciSpot, and job waiting times in conventional HPC deployments. To obtain the job waiting times in HPC clusters, we use the LANL Mustang traces published as part of the Atlas trace repository [16]. We analyze the waiting time of over two million jobs submitted over a 5 year period, and compute the increase in running time of the job due to the job waiting or queuing time.

Figure 7 compares the overhead (as percentage increase in running time) of SciSpot and HPC clusters for jobs of different lengths. We see that the average performance overhead due to waiting can be significant in the case of HPC clusters, and the job submission latency and queuing time dominate for smaller jobs, increasing their total turnaround time by more 2.5×. This waiting is amortized in the case of longer running jobs, and the overhead for longer jobs is around 30%.

On the other hand, SciSpot's performance overhead is significantly smaller for jobs of up to 8 hours in length. For longer jobs, the limited lifetime of Google Preemptible VMs (24 hours) begins to significantly increase the preemption probability and expected recomputation time. We emphasize

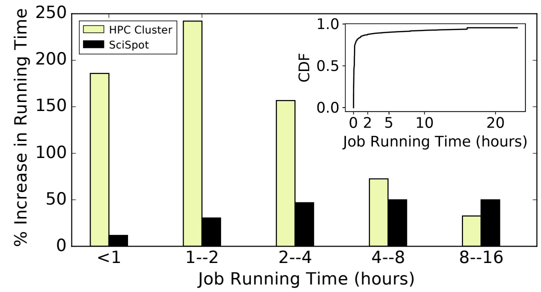

Figure 7: Increase in running time due to waiting on HPC clusters is significantly higher than the recomputation time for SciSpot, except for very long and rare jobs (see inset).

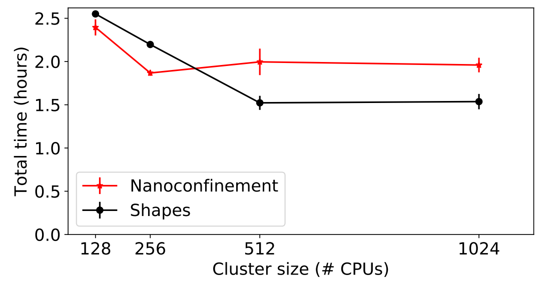

Figure 8: Bag of jobs running times exhibit classic parallel scaling behavior—performance improves until reaching a saturation point.

that these are *individual* job lengths, and not the running time of entire bag of jobs. We note that these large single jobs are rare, accounting for less than 5% of all HPC jobs (see inset in Figure 7). For smaller jobs (within a much larger bag), both the preemption probability and recomputation overhead is much smaller.

Result: *SciSpot's overhead of recomputation due to preemptions is small, and is up to* 10× *lower compared to the overhead of waiting in conventional HPC clusters.*

6.2 SciSpot Scaling

We now turn our attention to SciSpot's scaling properties. We are primarily interested in observing the behavior of running bags of jobs of different applications with different resource requirements. In all cases unless otherwise stated, we run bags of 36 jobs, and impose that 90% of all jobs complete (thus we target a completion of 32 jobs). The jobs in the bags are for exploring the different parameters (i.e., doing a parameter sweep), using SciSpot's automated parameter sweeping functionality described in Section 5. In the rest of this section, we evaluate SciSpot with different cluster sizes, number of preemptions, and bag sizes.

6.2.1 Increasing Cluster Size. It is common to deploy scientific computing applications on large clusters, and we evaluate SciSpot on different cluster sizes in Figure 8. The figure

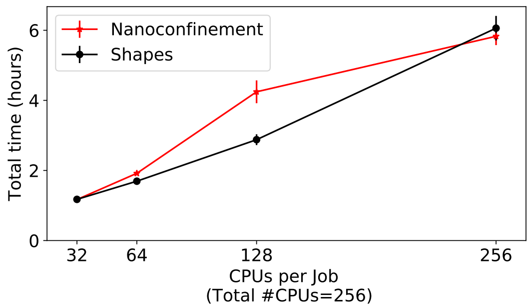

Figure 9: SciSpot can alleviate poor scaling by running more jobs in parallel and thus decreasing the intra-job parallelism (and hence number of CPUs per job, as shown in the figure).

shows the total running time (i.e., turnaround time) of the bag of jobs for the Nanoconfinement and Shapes applications as the total number of VMs (and hence total number of CPUs) increases. For this experiment, we used `n1-highcpu-32` VMs with 32 CPUs each, and we ran four jobs in parallel on the entire cluster. We see classic scaling behavior: both applications can scale to a higher number of VMs up to a point, after which communication overhead dominates, the performance saturates, and we see no reduction in running time.

We note that SciSpot provides the option to alleviate the parallel scaling bottleneck, by increasing the number of parallel jobs. That is, for a given fixed cluster size, it can run more jobs in parallel, by reducing the total resources allocated and hence the parallelism of an individual job. The effect of changing the number of parallel jobs is shown in Figure 9, which shows the running time of an entire bag of jobs when the total cluster size is fixed (256 CPUs), but the number of parallel jobs and hence the number of CPUs per job changes. We see that a *smaller* number of CPUs per job limits the communication overhead, and thus reduces the total running time of the bag. For both the Nanoconfinement and Shapes application, we see up to 6× reduction in the total bag running time when more number of jobs are launched in parallel and a smaller number of CPUs per job are used.

6.2.2 Increasing Bag Size. We now evaluate SciSpot's behavior when running larger bags of jobs. Table 1 shows the total running time of bags of 32 and 100 jobs. Since SciSpot reuses VMs when running jobs from a bag, it is able to take advantage of the relatively low preemption rates of VMs once they pass the first phase of early failures (Figure 2), and thus minimizes the number of preemptions as well as job failures. This makes SciSpot particularly suitable for running the large bags of jobs that are required when using machine learning techniques for HPC workloads [18, 22, 30, 31, 46, 52, 53, 60, 72], since the training and testing data needed for statistical machine learning can be generated using SciSpot's bag of jobs.

Application	Jobs	Time (Hours)	# Preemptions
Nanoconfinement	32	1.87	0
	100	6.08	1
Shapes	32	1.47	0
	100	4.49	5

Table 1: Running times and number of preemptions for bags of different sizes.

6.2.3 Increasing Preemptions. By reusing VMs across a bag of jobs and taking advantage of the low preemption rates during the middle of the 24 hour life of the preemptible VMs, the *expected* job failure rates and recomputation times are fairly small with SciSpot (as shown in Figures 5, 7). However, preemption rates can increase when the cloud operator sees high demand for resources. Figure 10 shows the running time of the bag of 32 Nanoconfinement jobs on a cluster of 4 `n1-highcpu-32` VMs, when different number of VMs are preempted. We see that even with a high number of preemptions, the running time only increases by 50%. This is because most job failures are due to early VM preemptions, as observed in our empirical and analytical models, and this reduces the recomputation time. We note that a higher than expected preemption rate (as shown in the figure) is rare, and happens with a vanishingly small likelihood. This shows that SciSpot is robust and can provide acceptable performance even under extreme, adverse conditions.

7 RELATED WORK

SciSpot builds upon a large body of prior work on running scientific computing applications on the cloud, and the various facets of transient cloud computing.

7.1 Cloud Computing For Science

Running scientific applications on the cloud introduces many tradeoffs compared to conventional HPC clusters, along the dimensions of performance, cost, scalability, convenience, and reproducability. These tradeoffs are explored in [20, 32, 37, 55, 80**?**]. In general, clouds can provide increased elasticity, lower waiting times, and more choices in resource allocation that can be tailored to the application. The cloud resource model is also present in platforms like nanoHUB [50], that provide easy execution and dissemination of nanotechnology simulation applications. Outside of the bags of jobs execution model, price optimizations for scientific workflows in the cloud is discussed in [33]. While bags of tasks [70] are often used for parallel applications, SciSpot is the first to use the bags of *jobs* abstraction for efficient and effective use of transient servers.

7.2 Transient Cloud Computing

The challenges posed by Amazon EC2 spot instances, the first transient cloud servers, have received significant attention from both academia and industry [10]. Since spot instances are significantly cheaper than the equivalent on-demand servers, they are attractive for running preemption and delay tolerant batch jobs [23, 29, 42, 51, 67, 74, 78]. A crucial component of EC2 spot instances is their dynamic auction-based pricing, and choosing the "right" bid price to minimize cost and performance degradation is the focus of much of the past work on transient computing [35, 43, 56, 66, 69, 73, 75, 76, 79, 81, 82]. However, as explained in Section 3.1, it remains to be seen how Amazon's recent change [13] in the preemption model of spot instances affects prior work.

On the other hand, the effective use of transient resources provided by other cloud providers such as Google, Microsoft, Packet, and Alibaba largely remains unexplored. Ours is the first work that studies the preemption characteristics and addresses the challenges involved in running large-scale applications on the Google Preemptible VMs, and provides insights on the unique preemption dynamics, as explained in Section 3.

7.2.1 Preemption Mitigation. Effective use of transient servers usually entails the use of fault-tolerance techniques such as checkpointing [62], migration [64], and replication. In the context of HPC workloads, [34, 54, 68] develop checkpointing and bidding strategies for MPI applications running on EC2 spot instances, based on older spot pricing models. Periodic checkpointing [28] is not appropriate in our case because preemptions are not memoryless.

By treating bags of jobs as an execution unit, allowing some jobs to fail, and using insights from preemption models, we show that it is possible to reduce the recomputation times to acceptable levels even without the use of periodic checkpointing that imposes additional deployment and performance overheads. Our preemption model for Google preemptible VMs developed in Section 3 provides a novel characterization of bathtub shaped failure rates not captured by the classic Weibull distribution, and is distinct from prior efforts [26, 57].

7.2.2 Server Selection. Optimized server selection is an important problem in cloud computing, and especially for transient servers because of the cost-performance-preemption tradeoff involved. Similar to SciSpot, SpotOn [67] is also a batch computing service that selects servers based on job characteristics and failure rates of different EC2 spot VMs. However, it is restricted to individual, single-VM batch jobs, and its design is tied to EC2 spot instances. The state of the art transient server selection involves the use of multiple types of VMs [63], and selecting a heterogeneous cluster can reduce the likelihood of mass concurrent preemptions. However, since scientific computing applications are mostly synchronous, even a single failure affects the entire job, and heterogeneous clusters are not required, and are in fact, detrimental [63]. Server selection is important even outside of preemptible VMs—developing bayesian optimization and application performance model based search for the "best" cloud VM is an active research area [15, 77], but these techniques do not account for preemptions.

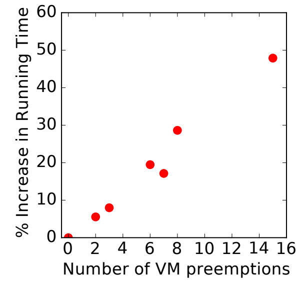

Figure 10: The increase in running time due to preemptions is under 50%, even when the number of preemptions is high.

8 CONCLUSION

Given the rise of transient cloud computing and its use in web services and distributed data processing, it is not the question of if, but when, transient cloud computing becomes a credible and powerful alternative to HPC clusters for scientific computing applications. In this paper, we developed principled approaches for deploying and orchestrating scientific computing applications on the cloud, and presented SciSpot, a framework for low-cost scientific computing on transient cloud servers. SciSpot develops the first empirical and analytical preemption model of Google Preemptible VMs, and uses the model for mitigating preemptions for "bags of jobs". SciSpot's cost-minimizing server selection and job scheduling policies can reduce costs by up to 5× compared to conventional cloud deployments. When compared to HPC clusters, SciSpot can reduce the total job turnaround times by up to 10×.

REFERENCES

[1] Alibaba Cloud Preemptible Instances. https://www.alibabacloud.com/help/doc-detail/52088.htm.
[2] Amazon EC2 Spot Instances, howpublished=https://aws.amazon.com/ec2/spot/,.
[3] Azure Low-priority Batch VMs. https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms.
[4] Big Red II at Indiana University. https://kb.iu.edu/d/bcqt.
[5] Google Cloud API Documentation. https://cloud.google.com/apis/docs/overview.
[6] Google Cloud Platform. https://cloud.google.com/.
[7] Google Cloud Preemptible VM Instances Documentation, howpublished=https://cloud.google.com/compute/docs/instances/preemptible,.
[8] Packet Spot Market. https://support.packet.com/kb/articles/spot-market.
[9] Slurm Workload Manager. https://slurm.schedmd.com/documentation.html.
[10] Spotinst. https://spotinst.com/.
[11] Torque Resource Manager. http://www.adaptivecomputing.com/products/torque/.
[12] Scientific Computing Using Spot Instances. http://aws.amazon.com/ec2/spot-and-science/, June 2013.

[13] New Amazon EC2 Spot pricing model. https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/, March 2018.

[14] Ali-Eldin, A., Westin, J., Wang, B., Sharma, P., and Shenoy, P. SpotWeb: Running Latency-sensitive Distributed Web Services on Transient Cloud Servers. In *HPDC* (2019).

[15] Alipourfard, O., and Yu, M. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. 15.

[16] Amvrosiadis, G., Park, J. W., Ganger, G. R., Gibson, G. A., Baseman, E., and DeBardeleben, N. On the diversity of cluster workloads and its impact on research results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 533–546.

[17] Ansel, J., Arya, K., and Cooperman, G. Dmtcp: Transparent checkpointing for cluster computations and the desktop. In *2009 IEEE International Symposium on Parallel & Distributed Processing* (2009), IEEE, pp. 1–12.

[18] Bartók, A. P., De, S., Poelking, C., Bernstein, N., Kermode, J. R., Csányi, G., and Ceriotti, M. Machine learning unifies the modeling of materials and molecules. *Science Advances 3*, 12 (2017).

[19] Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM TEC 1*, 3 (September 2013).

[20] Benedictis, A. D., Rak, M., Turtur, M., and Villano, U. Cloud-Aware Development of Scientific Applications. In *2014 IEEE 23rd International WETICE Conference* (Parma, Italy, June 2014), IEEE, pp. 149–154.

[21] Bougeret, M., Casanova, H., Rabie, M., Robert, Y., and Vivien, F. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[22] Ch'ng, K., Carrasquilla, J., Melko, R. G., and Khatami, E. Machine learning phases of strongly correlated fermions. *Phys. Rev. X 7* (Aug 2017), 031038.

[23] Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., and Krintz, C. See Spot Run: Using Spot Instances for MapReduce Workflows. In *HotCloud* (June 2010).

[24] Cirne, W., Brasileiro, F., Sauve, J., Andrade, N., Paranhos, D., Santos-neto, E., and Medeiros, R. Grid computing for bag of tasks applications. In *In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment* (2003), IFIP.

[25] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 153–167.

[26] Crevecoeur, G. A model for the integrity assessment of ageing repairable systems. *IEEE Transactions on reliability 42*, 1 (1993), 148–155.

[27] Daly, J. T. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems 22*, 3 (2006).

[28] Dongarra, J., Herault, T., and Robert, Y. Fault tolerance techniques for high-performance computing. 66.

[29] Dubois, D. J., and Casale, G. Optispot: minimizing application deployment cost using spot cloud resources. *Cluster Computing* (2016), 1–17.

[30] Ferguson, A. L. Machine learning and data science in soft materials engineering. *Journal of Physics: Condensed Matter 30*, 4 (2017), 043002.

[31] Fox, G., Glazier, J. A., Kadupitiya, J., Jadhao, V., Kim, M., Qiu, J., Sluka, J. P., Somogyi, E., Marathe, M., Adiga, A., et al. Learning Everywhere: Pervasive machine learning for effective High-Performance computation. *arXiv preprint arXiv:1902.10810* (2019).

[32] Galante, G., Erpen De Bona, L. C., Mury, A. R., Schulze, B., and da Rosa Righi, R. An Analysis of Public Clouds Elasticity in the Execution of Scientific Applications: a Survey. *Journal of Grid Computing 14*, 2 (June 2016), 193–216.

[33] GarÃŋ, Y., Monge, D. A., Mateos, C., and GarcÃŋa Garino, C. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing* (Feb. 2019).

[34] Gong, Y., He, B., and Zhou, A. C. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (Austin, Texas, 2015), ACM Press, pp. 1–12.

[35] Guo, W., Chen, K., Wu, Y., and Zheng, W. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15* (Portland, Oregon, USA, 2015), ACM Press, pp. 191–202.

[36] He, X., Shenoy, P., Sitaraman, R., and Irwin, D. Cutting the cost of hosting online services using cloud spot markets. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing* (2015), ACM, pp. 207–218.

[37] Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems 22*, 6 (June 2011), 931–945.

[38] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. Simulation of charged systems in heterogeneous dielectric media via a true energy functional. *Phys. Rev. Lett. 109* (Nov 2012), 223905.

[39] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity. *The Journal of Chemical Physics 138*, 5 (2013), 054119.

[40] Jadhao, V., Thomas, C. K., and Olvera de la Cruz, M. Electrostatics-driven shape transitions in soft shells. *Proceedings of the National Academy of Sciences 111*, 35 (2014), 12673–12678.

[41] Jadhao, V., Yao, Z., Thomas, C. K., and De La Cruz, M. O. Coulomb energy of uniformly charged spheroidal shell systems. *Physical Review E 91*, 3 (2015), 032305.

[42] Jain, N., Menache, I., and Shamir, O. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *11th International Conference on Autonomic Computing (ICAC 14)*, USENIX Association.

[43] Javadi, B., Thulasiram, R., and Buyya, R. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *UCC* (December 2011).

[44] Jing, Y., Jadhao, V., Zwanikken, J. W., and Olvera de la Cruz, M. Ionic structure in liquids confined by dielectric interfaces. *The Journal of chemical physics 143*, 19 (2015), 194508.

[45] Joaquim, P., Bravo, M., Rodrigues, L., and Matos, M. Hourglass: Leveraging transient resources for time-constrained graph processing in the cloud. In *Proceedings of the Fourteenth EuroSys Conference 2019* (New York, NY, USA, 2019), EuroSys '19, ACM, pp. 35:1–35:16.

[46] Kadupitiya, J., Fox, G., and Jadhao, V. Machine Learning for Performance Enhancement of Molecular Dynamics Simulations. In *Proceedings of the International Conference on Computational Science (ICCS) 2019* (2019), Springer.

[47] Kadupitiya, J., Marru, S., Fox, G. C., and Jadhao, V. Ions in nanoconfinement, Dec 2017. Online on nanoHUB; source code on GitHub at github.com/softmaterialslab/nanoconfinement-md.

[48] Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)* (Boston, USA, May 2013).

[49] Karlin, I., Keasler, J., and Neely, R. Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973, August 2013.

[50] Klimeck, G., McLennan, M., Lundstrom, M. S., and Adams III, G. B. nanohub. org-online simulation and more materials for semiconductors and nanoelectronics in education and research, 2008.

[51] Liu, H. Cutting MapReduce Cost with Spot Market. In *HotCloud* (June 2011).

[52] Liu, J., Qi, Y., Meng, Z. Y., and Fu, L. Self-learning monte carlo method. *Phys. Rev. B 95* (Jan 2017), 041101.

[53] Long, A. W., Zhang, J., Granick, S., and Ferguson, A. L. Machine learning assembly landscapes from particle tracking data. *Soft Matter 11*, 41 (2015), 8141–8153.

[54] Marathe, A., Harris, R., Lowenthal, D., De Supinski, B. R., Rountree, B., and Schulz, M. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC* (2014), ACM.

[55] Marathe, A., Harris, R., Lowenthal, D. K., De Supinski, B. R., Rountree, B., Schulz, M., and Yuan, X. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing* (2013), ACM, pp. 239–250.

[56] Mihailescu, M., and Teo, Y. M. The Impact of User Rationality in Federated Clouds. In *CCGrid* (2012).

[57] Mudholkar, G. S., and Srivastava, D. K. Exponentiated weibull family for analyzing bathtub failure-rate data. *IEEE transactions on reliability 42*, 2 (1993), 299–302.

[58] Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv. 51*, 1 (Jan. 2018), 8:1–8:29.

[59] Ouyang, X., Irwin, D., and Shenoy, P. Spotlight: An information service for the cloud. In *IEEE International Conference on Distributed Computing Systems (ICDCS)* (2016).

[60] Schoenholz, S. S. Combining machine learning and physics to understand glassy systems. *Journal of Physics: Conference Series 1036*, 1 (2018), 012021.

[61] Sharma, P. Transiency-driven Resource Management for Cloud Computing Platforms. https://scholarworks.umass.edu/dissertations_2/1388/, 2018.

[62] Sharma, P., Guo, T., He, X., Irwin, D., and Shenoy, P. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys* (April 2016).

[63] Sharma, P., Irwin, D., and Shenoy, P. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems* (June 2017), vol. 1, p. 23.

[64] Sharma, P., Lee, S., Guo, T., Irwin, D., and Shenoy, P. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys* (April 2015).

[65] Solis, F. J., Jadhao, V., and De La Cruz, M. O. Generating true minima in constrained variational formulations via modified lagrange multipliers. *Physical Review E 88*, 5 (2013), 053306.

[66] Song, Y., Zafer, M., and Lee, K. Optimal Bidding in Spot Instance Market. In *Infocom* (March 2012).

[67] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[68] Taifi, M., Shi, J. Y., and Khreishah, A. SpotMPI: A Framework for Auction-Based HPC Computing Using Amazon Spot Instances. In *Algorithms and Architectures for Parallel Processing*, Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7017. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 109–120.

[69] Tang, S., Yuan, J., and Li, X. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. In *CLOUD* (June 2012).

[70] Varshney, P., and Simmhan, Y. AutoBoT : Resilient and Cost-effective Scheduling of a Bag of Tasks on Spot VMs. *IEEE Transactions on Parallel and Distributed Systems* (2019), 1–1.

[71] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *EuroSys* (2015), ACM.

[72] Ward, L., Dunn, A., Faghaninia, A., Zimmermann, N. E., Bajaj, S., Wang, Q., Montoya, J., Chen, J., Bystrom, K., Dylla, M., et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science 152* (2018), 60–69.

[73] Wee, S. Debunking Real-Time Pricing in Cloud Computing. In *CCGrid* (May 2011).

[74] Wieder, A., Bhatotia, P., Post, A., and Rodrigues, R. Orchestrating the deployment of computations in the cloud with conductor. In *NSDI 12* (2012).

[75] Wolski, R., Brevik, J., Chard, R., and Chard, K. Probabilistic guarantees of execution duration for Amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17* (Denver, Colorado, 2017), ACM Press, pp. 1–11.

[76] Xu, H., and Li, B. A Study of Pricing for Cloud Resources. *Performance Evaluation Review 40*, 4 (March 2013).

[77] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17* (Santa Clara, California, 2017), ACM Press, pp. 452–465.

[78] Yi, S., Kondo, D., and Andrzejak, A. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (2010), IEEE, pp. 236–243.

[79] Zafer, M., Song, Y., and Lee, K. Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs. In *CLOUD* (2012).

[80] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[81] Zhang, Q., Gürses, E., Boutaba, R., and Xiao, J. Dynamic Resource Allocation for Spot Markets in Clouds. In *Hot-ICE* (March 2011).

[82] Zheng, L., Joe-Wong, C., Tan, C. W., Chiang, M., and Wang, X. How to Bid the Cloud. In *SIGCOMM* (August 2015).